UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
09/997,163	11/29/2001	Giuseppe Desoli	10011614-1	3624

22879 7590 04/08/2005

HEWLETT PACKARD COMPANY
P O BOX 272400, 3404 E. HARMONY ROAD
INTELLECTUAL PROPERTY ADMINISTRATION
FORT COLLINS, CO 80527-2400

EXAMINER	
PHAM, CHRYSTINE	
ART UNIT	**PAPER NUMBER**
2192	

DATE MAILED: 04/08/2005

Please find below and/or attached an Office communication concerning this application or proceeding.

Office Action Summary	Application No.	Applicant(s)	
	09/997,163	DESOLI ET AL.	
	Examiner	Art Unit	
	Chrystine Pham	2192	

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --

Period for Reply

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE 3 MONTH(S) FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If the period for reply specified above is less than thirty (30) days, a reply within the statutory minimum of thirty (30) days will be considered timely.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133). Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

Status

1)☒ Responsive to communication(s) filed on *15 November 2004*.

2a)☒ This action is **FINAL**. 2b)☐ This action is non-final.

3)☐ Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

Disposition of Claims

4)☒ Claim(s) *1-9,15-21 and 23-40* is/are pending in the application.

4a) Of the above claim(s) _____ is/are withdrawn from consideration.

5)☐ Claim(s) _____ is/are allowed.

6)☒ Claim(s) *1-9,15-21 and 23-40* is/are rejected.

7)☐ Claim(s) _____ is/are objected to.

8)☐ Claim(s) _____ are subject to restriction and/or election requirement.

Application Papers

9)☐ The specification is objected to by the Examiner.

10)☐ The drawing(s) filed on _____ is/are: a)☐ accepted or b)☐ objected to by the Examiner.

Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).

Replacement drawing sheet(s) including the correction is required if the drawing(s) is objected to. See 37 CFR 1.121(d).

11)☐ The oath or declaration is objected to by the Examiner. Note the attached Office Action or form PTO-152.

Priority under 35 U.S.C. § 119

12)☐ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).

a)☐ All b)☐ Some * c)☐ None of:

1.☐ Certified copies of the priority documents have been received.

2.☐ Certified copies of the priority documents have been received in Application No. _____.

3.☐ Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).

* See the attached detailed Office action for a list of the certified copies not received.

Attachment(s)

1) ☒ Notice of References Cited (PTO-892)
2) ☐ Notice of Draftsperson's Patent Drawing Review (PTO-948)
3) ☒ Information Disclosure Statement(s) (PTO-1449 or PTO/SB/08) Paper No(s)/Mail Date *1/18/2005*.
4) ☐ Interview Summary (PTO-413) Paper No(s)/Mail Date. _____ .
5) ☐ Notice of Informal Patent Application (PTO-152)
6) ☐ Other: _____.

DETAILED ACTION

1. This action is responsive to the Amendment filed on November 15th 2004. The Applicant has canceled claims 10-14, and 22. Claims 37-40 are new claims. Claims 1-9, 15-21, 23-40 are presented for examination.

Double Patenting

2. Claims 1-3, 5-7, 10, 12-15, 17-21, 23, and 30 have been provisionally rejected under the judicially created doctrine of obviousness-type double patenting as being unpatentable over claims 1, 8-10, 12-16, 18, 20-21, and 28 of copending Application No. 09999451 in view of Buzbee et al. (US 5933622), hereinafter, *Buzbee et al.* (see Office Action mailed August 26th 2004).

It is noted that the Applicant reserves the response to this rejection until such time if/when the provisional status of the rejection is lifted.

Claim Rejections - 35 USC § 112

3. The following is a quotation of the second paragraph of 35 U.S.C. 112:

The specification shall conclude with one or more claims particularly pointing out and distinctly claiming the subject matter which the applicant regards as his invention.

4. Claim 40 is rejected under 35 U.S.C. 112, second paragraph, as being indefinite for failing to particularly point out and distinctly claim the subject matter which applicant regards as the invention.

The term "substantially all" in claim 40 (line 2) is a relative term which renders the claim indefinite. The term "substantially all" is not defined by the claim, the specification does not provide a standard for ascertaining the requisite degree, and one of ordinary skill in the art would not be reasonably apprised of the scope of the invention. The office's rationale for viewing the term "substantially all" as relative term is such that according to a measuring system, the term "substantially all" [execution of a program] can be used to define an execution of a program which

has reached at least 52 percent of the code contained therein. According to yet another measuring system, the same term can be used to define an execution of a program which has reached at least 95 percent of the code contained therein. Furthermore, the definition of "substantially all" according to one system which uses the 95 percent mark differs greatly from the definition of the same term according to another system which uses, for instance, a 95.99 percent mark in measuring the degree/amount to which a program is executed.

Response to Arguments

5. Applicant's arguments filed November 15th 2004 have been fully considered but they are not persuasive.

In response to applicant's argument with respect to claims 4, 8-9, 11 which have been rejected under 35 U.S.C. 103(a) as being unpatentable over *Buzbee et al.* in view of Lethin et al. (*Lethin et al.*, US 6463582) that "neither reference provides a suggestion or motivation to modify the Buzbee system to include a just-in-time compiler", the Examiner recognizes that obviousness can only be established by combining or modifying the teachings of the prior art to produce the claimed invention where there is some teaching, suggestion, or motivation to do so found either in the references themselves or in the knowledge generally available to one of ordinary skill in the art. See *In re Fine*, 837 F.2d 1071, 5 USPQ2d 1596 (Fed. Cir. 1988)and *In re Jones*, 958 F.2d 347, 21 USPQ2d 1941 (Fed. Cir. 1992). In this case, both references have suggested the motivation (which has been cited in claim 4, see Office Action mailed August 26th 2004) to combine for the inclusion of the just-in-time compiler (see *translating code, different instruction set, original source code, hardware, new computer* col.1:10-45 of *Buzbee et al.*; see *dynamic object code conversion method, host processor* col.1:5-40 of *Lethin et al.*; see *host processor, target processor* Abstract of *Lethin et al.*). It is further noted that suggestion or motivation for the inclusion of the just-in-time compiler can also found in the knowledge generally available to one of ordinary skill in the art at the time of the Applicant's invention (see *just-in-time compiler* http://whatis.techtarget.com/definition/0,,sid9_gci212423,00.html).

In response to applicant's argument with respect to claims 5, 12, and 30 which have been rejected under 35 U.S.C. 103(a) as being unpatentable over *Buzbee et al.* in view of Challenger et al. (*Challenger et al.*, US 6256712) that "neither reference provides a suggestion or motivation to modify the Buzbee system to include an application programming interface for emitting translated program instructions into a code cache", the Examiner recognizes that obviousness can only be established by combining or modifying the teachings of the prior art to produce the claimed invention where there is some teaching, suggestion, or motivation to do so found either in the references themselves or in the knowledge generally available to one of ordinary skill in the art. See *In re Fine*, 837 F.2d 1071, 5 USPQ2d 1596 (Fed. Cir. 1988) and *In re Jones*, 958 F.2d 347, 21 USPQ2d 1941 (Fed. Cir. 1992). In this case, the motivation to combine the reference for the inclusion of the application programming interface (API) (which has been cited in claim 5, see Office Action mailed August 26th 2004) can be found in the knowledge generally available to one of ordinary skill in the art at the time of the Applicant's invention (see for example, *API* http://www.cnet.com/Resources/Info/Glossary/Terms/api.html; see *API* http://www.webopedia.com/TERM/A/API.html).

In response to applicant's argument with respect to claims 6 and 7 that "neither reference renders obvious 'interpreting and executing program instructions that have not be[en] emitted into the at least one code cache'... neither Buzbee nor Morely discloses 'interpreting' program instructions", the Examiner respectfully asks that the Applicant refer to the following definition of "interpreting" (program instructions) from Microsoft Computer Dictionary (Fifth Edition):

Interpret *vb.*

1. To translate a statement or instruction into executable form and then execute it.
2. To execute a program by translating one statement at a time into executable form and executing it before translating the next statement, rather than by translating the program

completely into executable code (compiling it) before executing it separately. See *also* interpreter. *Compare* compile.

According to the above definition, that is to say, the established meaning of "interpreting" in the art, the claimed limitation "interpreting program instructions" is clearly inherent in the dynamic translation system disclosed by *Buzbee et al.* (e.g., see *emulators, emulation system* col.1:25-col.2:35; see *dynamic translation system, run-time, executed, cache* col.2:64-col.3:20).

6. Applicant's arguments with respect to claims 1-9, 15-21, 23-40 have been considered but are moot in view of the following new ground(s) of rejection.

Claim Rejections - 35 USC § 103

7. The following is a quotation of 35 U.S.C. 103(a) which forms the basis for all obviousness rejections set forth in this Office action:

(a) A patent may not be obtained though the invention is not identically disclosed or described as set forth in section 102 of this title, if the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which said subject matter pertains. Patentability shall not be negatived by the manner in which the invention was made.

8. Claims 1-3, 5, 15, 17-19, 30-37 rejected under 35 U.S.C. 103(a) as being unpatentable over *Buzbee et al.* in view of *Challenger et al.*.

Claim 1

Buzbee et al. teach **a method** (e.g., col.1:5-10) **and system** (e.g., col.2:65) **for executing a program written for an original computer system** (e.g., see Abstract *first computer*) **on a different host computer system** (e.g., see Abstract *second computer*), comprising:

- **fetching original program instructions** (e.g., see *instruction, original code* col.1:28-50) **during execution of the program with an interpreter/emulator** (e.g., see *emulators, translate an instruction* col.1:28-35, see *dynamic translation system* col.2:65-col.3:35);

- **dynamically translating** (i.e., **interpreting) the original program instructions into instructions that can be executed by the host computer system** (e.g., see *instructions, new instruction set* col.1:28-50; see *dynamic translation system, blocks of code, sequence of instructions, trace, traces, target architecture,* col.2:65-col.3:35)
- **dynamically emitting translated program instructions into at least one code cache** (e.g., see *cache, executed* col.3:16-19) **of a virtual machine** (e.g., col.1:39-45); and
- **dynamically executing the translated instructions from the at least one code cache in lieu of the original program instructions when a semantic function of the original program instruction is requested** (e.g., see *cache, executed, source address, source code* col.3:14-35).

Buzbee et al. do not expressly disclose the at least one code cache of a dynamic execution layer interface of the virtual machine. However, *Challenger et al.* disclose **dynamically emitting data into the at least one code cache** (e.g., see *cache 2* FIG.1c & associated text) **via an application programming interface (API) of a dynamic execution layer interface** (e.g., see *cache manager 1, API's* FIG.1A & associated text) **that is associated with a translator** (e.g., see FIG.5 & associated text; see *compiler, run-time system* col.27:50-55). It would have been obvious to one of ordinary skill in the pertinent art at the time the invention was made to combine the teaching of *Buzbee et al.* with that of *Challenger et al.* to obtain a dynamic execution interface (i.e., API) which mediates between a translator and the at least one code cache since APIs, in general, are known to provide a level of abstraction between the application and the its functions/utilities/services to ensure portability/reusability of the code. Furthermore, APIs are designed to encapsulate complex implementation/coding of the API's functions, thus providing a streamlined interface to other applications/programs/software using the application. Other applications (e.g., a translator) can then utilize the application's services/functions (e.g., emit fragment, invalidate fragment) via invoking the methods exposed in the interface (e.g., cache manager API), without undue modification of the other applications' codes.

Claims 2-3

Claims recite limitations, which have been addressed in claim 1 (see **interpreter/emulator** in claim 1), therefore, are rejected for the same reasons as cited in claim 1.

Claim 5

Claim recites limitations, which have been addressed in claim 1 (see **emitting translated program instruction into at least one code cache, application programming interface** in claim 1), therefore, is rejected for the same reasons as cited in claim 1.

Claim 15

Buzbee et al. teach **an emulation program** (e.g., see *programs, translating code, different instruction sets* col.1:10-15; see *emulators, translate an instruction* col.1:28-35, see *dynamic translation system* col.2:65-col.3:35) **configured to emulate an original computer system** (e.g., see Abstract *first computer*) **for which a program was written, the emulation program stored on a computer-readable medium** (e.g., see *branch taking trap, trap handling routine* col.2:40-55) and comprising:

- **logic configured to dynamically translate (i.e., interpret) original program code into instructions that can be executed by a host computer system** (e.g., see *instructions, new instruction set* col.1:28-50; see *dynamic translation system, blocks of code, sequence of instructions, trace, traces, target architecture,* col.2:65-col.3:35; see Abstract *second computer*);
- **logic configured to dynamically emit code fragment translations of the original program code** (e.g., see *dynamic translation system, blocks of code, sequence of instructions, trace, traces, target architecture,* col.2:65-col.3:35;) **into at least one code cache** (e.g., see *cache, executed* col.3:16-19) **of a virtual machine** (e.g., col.1:39-45); and
- **logic configured to dynamically execute the code fragments within the at least one code cache in lieu of the original program code when a semantic function of the associated program code is requested** (e.g., see *cache, executed, source address, source code* col.3:14-35).

Buzbee et al. do not expressly disclose the at least one code cache of a dynamic execution layer interface of the virtual machine. However, *Challenger et al.* disclose **dynamically emitting data into the at least one code cache** (e.g., see *cache 2* FIG.1c & associated text) **via an application programming interface (API) of a dynamic execution layer interface** (e.g., see *cache manager 1, API's* FIG.1A & associated text) **that is associated with a translator** (e.g., see FIG.5 & associated text; see *compiler, run-time system* col.27:50-55). It would have been obvious to one of ordinary skill in the pertinent art at the time the invention was made to combine the teaching of *Buzbee et al.* with that of *Challenger et al.* to obtain a dynamic execution interface (i.e., API) which mediates between a translator and the at least one code cache since APIs, in general, are known to provide a level of abstraction between the application and the its functions/utilities/services to ensure portability/reusability of the code. Furthermore, APIs are designed to encapsulate complex implementation/coding of the API's functions, thus providing a streamlined interface to other applications/programs/software using the application. Other applications (e.g., translator) can then utilize the application's services/functions (e.g., emit fragment, invalidate fragment) via invoking the methods exposed in the interface (e.g., cache manager API), without undue modification of the other applications' codes.

Claim 17

Claim recites limitations, which have been addressed in claim 15 (see **emit code fragment translations, application programming interface** in claim 15), therefore, is rejected for the same reasons as cited in claim 15.

Claim 18

Claim recites limitations, which have been addressed in claim 15 (see **interpret, execute** in claim 15), therefore, is rejected for the same reasons as cited in claim 15.

Claim 19

The rejection of base claim 15 is incorporated. *Buzbee et al.* further teach **logic configured to emulate exception handling actions of the original computer system for which a program was written** (e.g., see *source machine, source code, asynchronous event handling, trap handling routine, execution, halted* col.2:65-col.4:65).

Claim 30

Buzbee et al. teach a **translator/emulating system** comprising:

- **emit code fragment function with which the translator can emit code fragments** (e.g., see *dynamic translation system, blocks of code, sequence of instructions, trace, traces, target architecture,* col.2:65-col.3:35) **into code cache** (e.g., see *cache, executed* col.3:16-19); and
- an **execute function with which the translator can request execution of code fragments contained within the at least one code cache** (e.g., see *cache, executed* col.3:14-35).

Buzbee et al.- do not expressly disclose an application programming interface configured to link a translator to a dynamic execution interface of the emulating system, comprising: an emit fragment function with which the translator can emit code fragments into code caches of the dynamic execution layer interface; and an execute function with which the translator can request execution of code fragments contained within the at least one code cache. However, *Challenger et al.* disclose **emitting data into the at least one code cache** (e.g., see *cache 2* FIG.1A & associated text) **via an application programming interface (API) of a dynamic execution layer interface** (e.g., see *cache manager 1, API's* FIG.1A & associated text) **that is linked to a translator** (e.g., see FIG.5 & associated text; see *compiler, run-time system* col.27:50-55). *Challenger et al.* further teach

- **the emit fragment function** (e.g., see *410* FIG.4 & associated text) **can be used to perform at least one of tracking a cached code fragment and associating metadata** (e.g., see *object_id* FIG.4 & associated text) **with a cached code fragment.**
- **a lookup fragment function with which cached code fragments can be retrieved** (e.g., see *415* FIG.4 & associated text).

- **an invalidate fragment function with which individual cached code fragments can be invalidated** (e.g., see *450* FIG.4 & associated text).
- **a cache flush function in which caches of the dynamic execution layer interface can be flushed** (e.g., col.1:30-35, col.5:41-42, see *410* FIG.4 & associated text).
- **an install callback function with which the translator can be notified as to particular events that occur within the dynamic execution layer interface** (e.g., see *return status 1240* FIG.6 & associated text, see *1680* FIG.7 & associated text, see *1150* FIG.8 & associated text, see FIG.9-11, 17 & associated text).
- **a enumerate fragment function with which cached code fragments can be numerated using associated metadata** (e.g., see *470* FIG.4 & associated text).

It would have been obvious to one of ordinary skill in the pertinent art at the time the invention was made to combine the teaching of *Buzbee et al.* with that of *Challenger et al.* to obtain a application programming interface (API) configured to link a translator to a dynamic execution layer interface to enable the translator to emit program code into the code cache and request execution of the program code, since APIs, in general, are known to provide a level of abstraction between the applications and the their functions/utilities/services to ensure portability/reusability of the code. Furthermore, APIs are designed to encapsulate complex implementation/coding of the API's functions (i.e., emit fragment function, cache flush function), thus providing a streamlined interface to other applications/programs/software using the application. Other applications (e.g., the translator) can then utilize the application's services/functions (e.g., emit fragment function, invalidate fragment function) via invoking the methods exposed in the interface (e.g., cache manager API), without undue modification of the other applications' codes.

Claim 31

The rejection of base claim 30 is incorporated. *Challenger et al.* further teach **the emit fragment function** (e.g., see *410* FIG.4 & associated text) **can be used to perform at least one of tracking a cached code fragment and associating metadata** (e.g., see *object_id* FIG.4 & associated text) **with a cached code fragment.**

Since the function is part of the API (i.e., cache manager) taught by *Challenger et al.*, the motivation to incorporate *Challenger et al.* into that of *Buzbee et al.* is the same, which has been cited in claim 30.

Claim 32

The rejection of base claim 30 is incorporated. *Challenger et al.* further teach **a lookup fragment function with which cached code fragments can be retrieved** (e.g., see *415* FIG.4 & associated text). Since the function is part of the API (i.e., cache manager) taught by *Challenger et al.*, the motivation to incorporate *Challenger et al.* into that of *Buzbee et al.* is the same, which has been cited in claim 30.

Claim 33

The rejection of base claim 30 is incorporated. *Challenger et al.* further teach **an invalidate fragment function with which individual cached code fragments can be invalidated** (e.g., see *450* FIG.4 & associated text). Since the function is part of the API (i.e., cache manager) taught by *Challenger et al.*, the motivation to incorporate *Challenger et al.* into that of *Buzbee et al.* is the same, which has been cited in claim 30.

Claim 34

The rejection of base claim 30 is incorporated. *Challenger et al.* further **teach a enumerate fragment function with which cached code fragments can be numerated using associated metadata** (e.g., see *470* FIG.4 & associated text). Since the function is part of the API (i.e., cache manager) taught by *Challenger et al.*, the motivation to incorporate *Challenger et al.* into that of *Buzbee et al.* is the same, which has been cited in claim 30.

Claim 35

The rejection of base claim 30 is incorporated. *Challenger et al.* further teach **a cache flush function in which caches of the dynamic execution layer interface can be flushed** (e.g., col.1:30-35, col.5:41-42, see *410*

FIG.4 & associated text). Since the function is part of the API (i.e., cache manager) taught by *Challenger et al.*, the motivation to incorporate *Challenger et al.* into that of *Buzbee et al.* is the same, which has been cited in claim 30.

Claim 36

The rejection of base claim 30 is incorporated. *Challenger et al.* further **teach an install callback function with which the translator can be notified as to particular events that occur within the dynamic execution layer interface** (e.g., see *return status 1240* FIG.6 & associated text, see *1680* FIG.7 & associated text, see *1150* FIG.8 & associated text, see FIG.9-11, 17 & associated text). Since the function is part of the API (i.e., cache manager) taught by *Challenger et al.*, the motivation to incorporate *Challenger et al.* into that of *Buzbee et al.* is the same, which has been cited in claim 30.

Claim 37

The rejection of base claim 1 is incorporated. *Buzbee et al.* further teach **accessing original memory addresses to identify actual locations of the instructions on the host computer system** (e.g., see *source address, target address, target architecture* col.2:65-col.3:35).

9. Claims 4, 8-9, 16, 20-29 are rejected under 35 U.S.C. 103(a) as being unpatentable over *Buzbee et al.* in view of *Challenger et al.*, further in view of *Lethin et al.*.

Claim 4

The rejection of base claim 1 is incorporated. *Buzbee et al.* further teach **dynamically translating the program instructions with an optimizing compiler** (e.g., see *compilers, optimizing compilers* col.4:40-65). *Buzbee et al.* do not expressly disclose the optimizing compiler as being just-in-time (JIT) compiler. However, *Lethin et al.* discloses **an optimizing compiler as being the JIT compiler/translator** (e.g., see Abstract, FIG.24,25 & associated text). *Lethin et al.* further disclose the **JIT compiler growing a code segment by linking**

program instructions together prior to emitting translated program code (e.g., col.2:10-15, see FIG.2,8,9,10 & associated text). It would have been obvious to one of ordinary skill in the pertinent art at the time the invention was made to modify the teaching of *Buzbee et al.* to include the JIT compiler as disclosed by *Lethin et al.*, which would produce the expected result with reasonable success. And the motivation for doing so would have been that traditional compilers would need to obtain and compile the source code to produce the corresponding executable code. The downside of traditional compilers is that the source code might not be obtainable to a new computer system with a different instruction set, and traditional compilers produce executable code which is platform-dependent. JIT compilers would have been preferred instead as being capable of determining program methods/instructions which are called most often, interpreting and translating the associated byte code at runtime to produce platform-independent executable code for the instructions.

Claims 8-9, 16

Claims recite limitations, which have been addressed in claim 4 (see **JIT compiler, growing a code segment by linking program instructions together prior to emitting translated program code** in claim 4), therefore, are rejected for the same reasons as cited in claim 4. Since the claimed feature is part of the JIT compiler disclosed by *Lethin et al.*, motivation for incorporating *Lethin et al.* into *Buzbee et al.* is the same, which has been cited in claim 4.

Claim 20

Buzbee et al. teach **a system** (e.g., col.2:65) **for executing program code that was written for an original computer system** (e.g., see Abstract *first computer*) **on a different host computer system** (e.g., see Abstract *second computer*), comprising:

- **an emulator** (e.g., see *emulators, translate an instruction* col.1:28-35) **configured to fetch original program instructions during execution of a program** (e.g., see *instruction, original code* col.1:28-50; see *dynamic translation system* col.2:65-col.3:35);

- **a compiler** (e.g., see *compilers, optimizing compilers* col.4:40-65) **configured to dynamically translate the original program instructions into instructions that can be executed by the host computer system** (e.g., see *instructions, new instruction set* col.1:28-50; see *dynamic translation system, blocks of code, sequence of instructions, trace, traces, target architecture,* col.2:65-col.3:35)
- **a virtual machine** (e.g., col.1:39-45) **including a core having at least one code cache in which code fragments can be dynamically cached and executed** (e.g., see *cache, executed* col.3:14-35); and

Buzbee et al. do not expressly disclose the compiler being a just-in-time compiler. However, *Lethin et al.* disclose a just-in-time compiler (e.g., see Abstract, FIG.24,25 & associated text). However, *Lethin et al.* discloses **an optimizing compiler as being the JIT compiler/translator** (e.g., see Abstract, FIG.24,25 & associated text). *Lethin et al.* further disclose the **JIT compiler growing a code segment by linking program instructions together prior to emitting translated program code** (e.g., col.2:10-15, see FIG.2,8,9,10 & associated text). It would have been obvious to one of ordinary skill in the pertinent art at the time the invention was made to modify the teaching of *Buzbee et al.* to include the JIT compiler as disclosed by *Lethin et al.*, which would produce the expected result with reasonable success. And the motivation for doing so would have been that traditional compilers would need to obtain and compile the source code to produce the corresponding executable code. The downside of traditional compilers is that the source code might not be obtainable to a new computer system with a different instruction set, and traditional compilers produce executable code which is platform-dependent. JIT compilers would have been preferred instead as being capable of determining program methods/instructions which are called most often, interpreting and translating the associated byte code at runtime to produce platform-independent executable code for the instructions.

Buzbee et al. do not expressly disclose the virtual machine comprising a dynamic execution layer interface and an application programming interface that links the translator to the virtual machine. However, *Challenger et al.* disclose **dynamically emitting data into the at least one code cache** (e.g., see *cache 2* FIG.1c & associated text) **via an application programming interface (API) of a dynamic execution layer interface**

(e.g., see *cache manager 1, API's* FIG.1A & associated text) **that is associated with a translator** (e.g., see FIG.5 & associated text; see *compiler, run-time system* col.27:50-55). It would have been obvious to one of ordinary skill in the pertinent art at the time the invention was made to combine the teaching of *Buzbee et al.* with that of *Challenger et al.* to obtain a dynamic execution interface (i.e., API) which mediates between a translator and the at least one code cache since APIs, in general, are known to provide a level of abstraction between the application and the its functions/utilities/services to ensure portability/reusability of the code. Furthermore, APIs are designed to encapsulate complex implementation/coding of the API's functions, thus providing a streamlined interface to other applications/programs/software using the application. Other applications (e.g., a translator) can then utilize the application's services/functions (e.g., emit fragment, invalidate fragment) via invoking the methods exposed in the interface (e.g., cache manager API), without undue modification of the other applications' codes.

Claim 21

The rejection of base claim 20 is incorporated. Claim recites limitations, which have been addressed in claim 1 (see **interpreter/emulator** in claim 1), therefore, is rejected for the same reasons as cited in claim 1.

Claim 23

The rejection of base claim 20 is incorporated. Claim recites limitations, which have been addressed in claim 30, therefore, is rejected for the same reasons as cited in claim 30.

Claims 24-29

Claims recite limitations, which have been addressed in claims 31-36, therefore, are rejected for the same reasons as cited in claims 31-36.

10. Claims 6-7 are rejected under 35 U.S.C. 103(a) as being unpatentable over *Buzbee et al.* in view of *Challenger et al.* further in view of Morley of record, hereinafter, *Morley*.

Claim 6

The rejection of base claim 1 is incorporated. *Buzbee et al.* further teach the step of **interpreting program instructions that have not been emitted into the at least one code cache** (see **interpreting, emitting translated program instructions** in claim 1). *Buzbee et al.* do not expressly disclose the step of executing program code that has not been emitted into the at least one code cache. However, *Morley* **discloses an emulation method and program** (e.g., col.1:5-12, col.3:32-33) **for emulating an original computer system for which a program was written** (e.g., see Abstract, col.3:33-35), **the emulation program stored on computer-readable medium** (e.g., col.3:37-44), **the method/program comprising the step of executing program code that has not been emitted into the at least one code cache** (e.g., col.1:44-49 & 60-65, col.3:65-col.4:6). It would have been obvious to one of ordinary skill in the pertinent art at the time the invention was made to combine that teaching of *Morley* with that of *Buzbee et al.* to include the step of executing program code that has not been emitted into the at least one code cache which would produce the expected result with reasonable success. And the motivation for doing so would have been that the capability to execute program code which are not emitted to code caches allows programmers/designers of the emulation system the versatility of storing program code in various different types of memory (e.g., main memory, cache memory, and/or permanent storage) within the emulation system depending on their choices and/or the particular architecture of said emulation system.

Claim 7

The rejection of base claim 6 is incorporated. *Buzbee et al.* further teach **emulating exception handling actions that would have been performed by the original computer system during execution** (e.g., see *source machine, source code, asynchronous event handling, trap handling routine, execution, halted* col.2:65-col.4:65).

11. Claims 38-40 are rejected under 35 U.S.C. 103(a) as being unpatentable over *Buzbee et al.* in view of *Challenger et al.* further in view of Shenoy et al. (US 5455924), hereinafter, *Shenoy et al.*.

Claim 38

The rejection of base claim 1 is incorporated. *Buzbee et al.* do not expressly disclose executing translated instructions within the at least one code cache until a cache miss occurs. However, *Shenoy et al.* disclose **executing translated instructions within the at least one code cache until a cache miss occurs** (e.g., see *cache, block, execution, store instructions, load instruction* Abstract; see *program execution, cache memory unit, high speed cache, "cache miss"* col.1:34-col.2:48; see *instruction execution, store instruction, "stalling"* col.5:9-40). It would have been obvious to one of ordinary skill in the pertinent art at the time the invention was made to incorporate the teaching of *Shenoy et al.* into that of *Buzbee et al.* for the inclusion of executing translated instructions within the at least one code cache until a cache miss occurs. And the motivation for doing so would have been to provide consistency within the memory structure and cache unit of computer system. For instance, if an unfinished load instruction is on the instruction queue, then the desired contents of the address for the load instruction have not yet been read from external memory. If a subsequent store instruction is allowed to executed before the missed load instruction completes then the store instruction may overwrite the desired data in the external memory (with the data associated with the store instruction at the same address location) before the data has a chance to be read from the external memory for the load instruction (e.g., see *Shenoy et al.* col.5:9-40).

Claim 39

The rejection of base claim 38 is incorporated. Claim recites limitations, which have been addressed in claim 1, therefore, is rejected for the same reasons as cited in claim 1.

Claim 40

The rejection of base claim 39 is incorporated. *Buzbee et al.* further teach **repeating the procedures of claim 39 until substantially all execution of the program comprises execution of translated program instructions within the at least one code cache** (e.g., see *running of source program, first computer, second computer* Abstract).

Conclusion

12. The prior art made of record and not relied upon is considered pertinent to applicant's disclosure.

13. Applicant's amendment necessitated the new ground(s) of rejection presented in this Office action. Accordingly, **THIS ACTION IS MADE FINAL**. See MPEP § 706.07(a). Applicant is reminded of the extension of time policy as set forth in 37 CFR 1.136(a).

A shortened statutory period for reply to this final action is set to expire THREE MONTHS from the mailing date of this action. In the event a first reply is filed within TWO MONTHS of the mailing date of this final action and the advisory action is not mailed until after the end of the THREE-MONTH shortened statutory period, then the shortened statutory period will expire on the date the advisory action is mailed, and any extension fee pursuant to 37 CFR 1.136(a) will be calculated from the mailing date of the advisory action. In no event, however, will the statutory period for reply expire later than SIX MONTHS from the date of this final action.

14. Any inquiry concerning this communication or earlier communications from the examiner should be directed to Chrystine Pham whose telephone number is 571-272-3702. The examiner can normally be reached on Mon-Fri, 8:30am-5pm.

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, Tuan Q Dam can be reached on 571-272-3695. The fax phone number for the organization where this application or proceeding is assigned is 703-872-9306.

Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system. Status information for published applications may be obtained from either Private PAIR or Public PAIR. Status information for unpublished applications is available through Private PAIR only. For more information about the PAIR system, see http://pair-direct.uspto.gov. Should you have questions on access to the Private PAIR system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free).

CP
March 28, 2005

TUAN DAM
SUPERVISORY PATENT EXAMINER